2Б

УДК 681.3

М.К. Буза
Белорусский государственный университет, г. Минск, Беларусь
Беларусь, г.Минск, пр-т Независимости 4, *bouza@bsu.by*

Среда проектирования распределенной обработки данных

M.K. Bouza
Belarusian state University, Minsk, Belarus
Belarus, Minsk, pr-t Independence 4, bouza@bsu.by

Environment Design of Distributed Data Processing

М.К. Буза
Білоруський державний університет, м. Мінськ, Білорусь
Білорусь, м. Мінськ, пр. Незалежності 4, *bouza@bsu.by*

Середовище проектування розподіленої обробки даних

Рассмотрен и проанализирован ряд существующих концепций распределенной обработки данных. Сформулированы основные требования, предъявляемые к таким системам, и предложены стратегии их реализации. Выбрана и обоснована технология с открытым кодом, поддерживающая распределенную обработку. Расширены ее функциональные возможности по обработке больших объемов данных, идентификации и обработке сбоев рабочих процессов.
Ключевые слова: распределенная обработка, стратегии реализации, открытый код, масштабируемость, идентификация, пакетирование

Reviewed and analyzed a number of existing concepts of distributed data processing. The basic requirements to such systems and strategies for their implementation was suggested. Selected and proved technology of open source that supports distributed processing. Expanded its functional capabilities for processing large amounts of data, identification and handling failures workflows.
Keywords: distributed processing, implementation strategies, open source, scalable, identification, packaging

Розглянуто і проаналізовано ряд існуючих концепцій розподіленої обробки даних. Сформульовані основні вимоги, що пред'являються до таких систем, та запропоновано стратегії їх реалізації. Обрана і обґрунтована технологія з відкритим кодом, що підтримує розподілену обробку. Розширені її функціональні можливості по обробці великих обсягів даних, ідентифікації та обробки збоїв робочих процесів.
Ключові слова: розподілена обробка, стратегії реалізації, відкритий код, масштабованість, ідентифікація, пакетування

Введение

На сегодняшний день накоплен огромный объем различных данных, которые необходимо обработать за разумное время. Как сказал Эрик Шмидт из Google: «С момента зарождения цивилизации и до 2003г. человечество накопило пять экзабайт данных. Сейчас такой объем мы генерируем за два дня».

Огромный объем данных генерируют телефонные звонки, сайты социальных сетей, вообще пользователи Интернета. Мы всюду оставляем след своих действий, создавая информацию.

Теперь, пожалуй, обработка больших данных занимает позицию как тренд номер два в информационно-технологической инфраструктуре после виртуализации.

С другой стороны, исследователи приступают к решению сложных задач, требующих высокоэффективных вычислительных платформ. Среди них выделяются задачи климатологии, моделирования в физике, поиск внеземных цивилизаций [1].

Для решения таких задач за ограниченное время требуется создавать системы распределенной обработки данных (РОД). Однако здесь возникают новые проблемы: динамичное управление ресурсами, оптимизация организации распределенной памяти, минимизация взаимодействий между различными процессами, обеспечение отказоустойчивости всей системы и другие [2].

Отсюда следует необходимость создания системы распределенной обработки, минимизирующей влияние некоторых из перечисленных проблем.

Цель работы – расширение функций РОД по динамичному управлению процессами, обнаружению и обработке отказов рабочих процессов.

Ниже предлагается масштабируемая система распределенной обработки, выполняющая параллельные вычисления и обрабатывающая отказы ее узлов.

Выбор базовой технологии

Среда распределенных систем ассоциируется как с Grid-технологиями, так и с облачными вычислениями. Несмотря на то, что инфраструктура «облаков» существует в различных формах, фундаментальная основа ее состоит в том, что любой пользователь компьютерных систем может через интернет получить необходимые ему информационные услуги. При этом требуется решить две важнейшие проблемы: масштабирование и обработка отказов в течение всего времени работы.

Важнейшей составляющей облачных вычислений является система сборки результатов. Интегрированные среды типа Visual Studio и Eclipse, актуальные в 2000-х годах, сменились современными встроенными предметно-ориентированными языками, такими как Gradle, SCons, Rake и Shake.

В качестве основы для реализации указанных задач было выбрано решение с открытым кодом Terracotta, обеспечивающее практически линейное масштабирование и высокую надежность [3]. Кроме того, Terracotta поддерживает стандартную HTTP кластеризацию сессий в серверах Apache Tomcat и Oracle WebLogic, а также интегрируется с другими проектами с открытым кодом: Struts, Spring, Hibernate.

Во время кластеризации решаются две важнейших задачи – масштабирование и обработка отказов. Причем при вертикальной кластеризации осуществляется распределение нагрузки на разных экземплярах сервера, реально функционирующих на одном физическом сервере. Горизонтальная кластеризация осуществляет распределение нагрузки среди множества обработчиков. Решаемые кластеризацией задачи очень важны для систем, которым необходимо обеспечение безотказной работы в течение всего времени функционирования. Здесь важно выбрать корректную поведенческую стратегию системы в случае отказа основного сервера, включающую репликацию состояния пользователя и перенаправление его запросов на один из вспомогательных серверов.

Посредством конфигурационного файла задаются основной и вспомогательный сервера. Один из конфигурационных файлов может быть представлен следующим образом

```
<tc : tc - config
xsi : schemalocation = ″http: // www. terracotta.org/shema/terracotta-5.xsd″
xmlns : tc = ″http: // www. terracotta.org/config″
xmlns : xsi = ″http: // www.w3.org/2001/XMLSchena-instance″>
…
<server host = ″123.456.7.890″ name = ″Server1″>
…
<server host = ″myResolvableHostName″ name = ″Server2″>
…
</ tc : tc - config>
```

Сам процесс пересылки данных осуществляется через конвертацию Java-объектов в бинарные объекты, называемую сериализацией. Стандартная схема этой акции состоит в определении измененных объектов, их конвертации и отсылке результата на реплицированный сервер.

Будем использовать bytecode instrumentation (BCI) на серверах и клиентах для определения измененных объектов и реплицировать их только в кластере. BCI по сути является механизмом изменения поведения приложения, работающего в реальном времени. Достигается оно через перехват события изменения состояния объекта, модифицируя его bytecode.

При создании обработчика отказов необходимо решить вопрос, когда и куда пересылать кластерную информацию. Стандартный подход состоит в пересылке реплицированных данных всем серверам системы сразу же после изменения [4]. Но это порождает большую избыточность. Лучшее решение – отсылать измененные данные только некоторым серверам, указанным в конфигурационном файле, а остальные могут получить эту информацию по требованию.

Используя решение Terracotta можно из любой многопоточной программы [2], изначально определенную для работы на одном узле, сделать многоузловую, не изменяя кода, и используя обычный Java синтаксис. Так как Terracotta является технологией с открытым кодом и поддерживает возможность разработки дополнительных модулей и их интеграцию, то используем эти возможности для функционального расширения ее возможностей

Модуль расширения для TERRACOTTA

Разработанный модуль расширения Terracotta базируется на шаблоне «мастер/рабочий», но имеет ряд преимуществ по сравнению с большинством стратегий реализации данного шаблона.

При разработке расширения были учтены проблемы и задачи, которые появляются во время реализации платформы для параллельных вычислений. В частности,

- большой объем данных;
- маршрутизация;
- сбой работы и обработка данного типа сбоя;
- динамичное управление ресурсами системы.

Диаграмма классов для предлагаемого расширения системы приведена на рис. 1.

Стандартная реализация шаблона «мастер/рабочий» состоит в реализации очереди задач между мастерами и рабочими. Однако, если мы решаем проблему обработки больших объемов данных, то одна очередь может стать узким местом в процессе вычислений.

Мы можем использовать многопоточные вычисления [5] или OpenCL [6], если выбранная технология ее поддерживает.

Одним из вариантов решения проблемы может стать введение отдельных очередей для каждого рабочего процесса, а функционал мастера расширить балансировкой нагрузки на данные очереди. Такой подход позволяет:

– увеличить использование локальных ссылок. Если все вычисления над одними и теми же данными направлены в одну очередь, то рабочие процессы не будут совместно использовать объекты, находящиеся в различных Java Virtual Machine (JVM). Изменение объекта не приведет к увеличению трафика в системе;

– уменьшить конкуренцию между рабочими процессами, ибо для каждой очереди будет один пишущий и один читающий субъект.

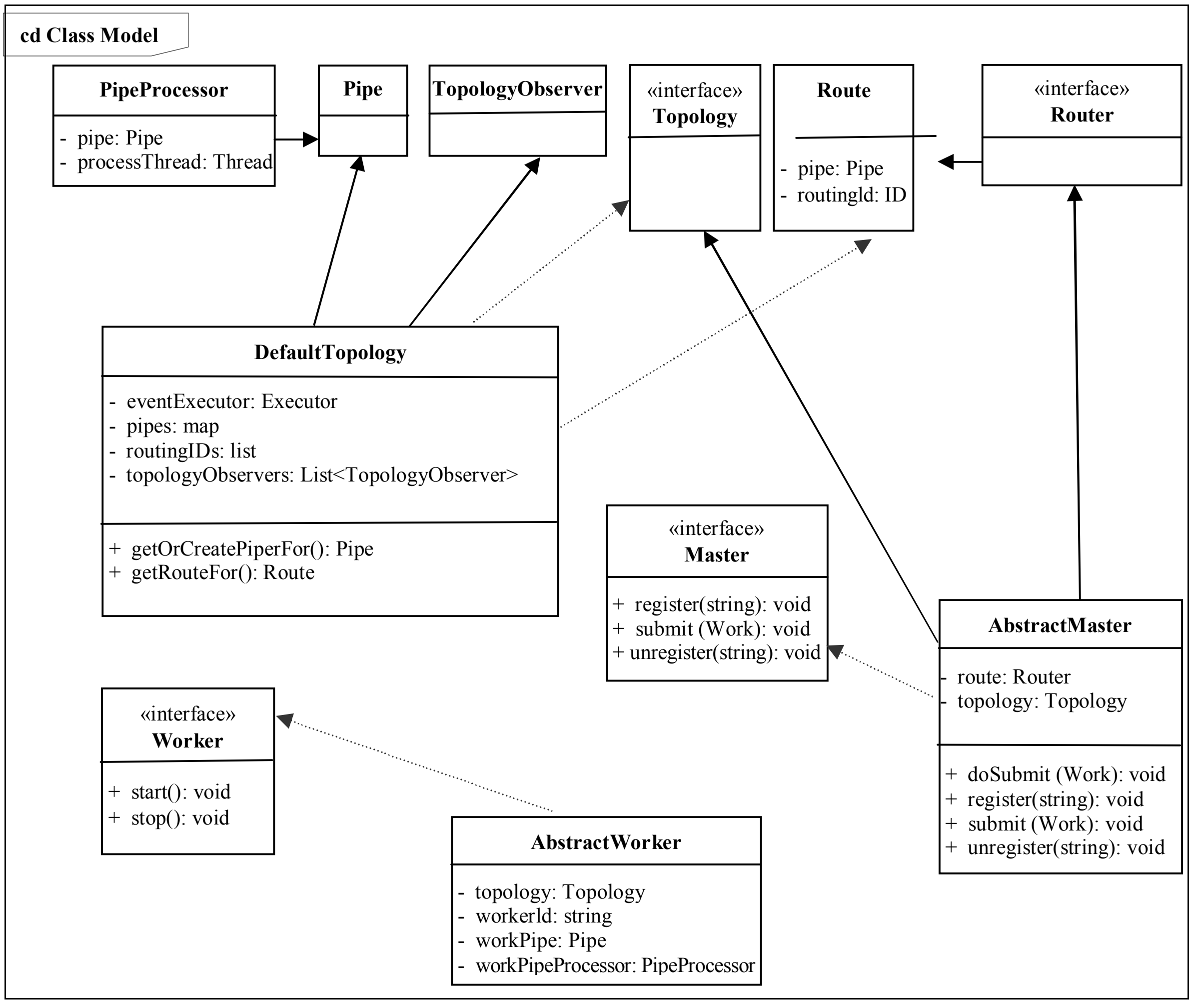


Рисунок 1 – Диаграмма основных классов модуля расширения

Более сильный вариант сокращения конкуренции – разбиение каждой из рабочих очередей на две. Одна очередь для отправки работы от мастера к рабочему, а вторая для отсылки полученных результатов. Реализация такого подхода в созданном модуле потребовала разработки класса-обертки над предложенными двумя очередями, включающая Pipe(канал) и класс PipeProcessor, отвечающих за управление различными каналами. Каждому каналу назначен уникальный номер и работы от мастера направляются в ту или иную очередь в зависимости от выбранного алгоритма маршрутизации и схемы обработки отказов. Реализованы четыре стратегии распределения работы по узлам:

– произвольный выбор узла;
– выбор узла с наименьшей загрузкой;
– направление на узел, где имеются необходимые для работы данные;
– стратегия карусели.

Реализация механизма пакетирования заданий и динамического управления рабочими процессами

Новые концепции, например, каналы с входящими и исходящими очередями, вспомогательные классы для управления маршрутами и каналами уникальные номера маршрутов для каждого рабочего недостаточны для стабильной работы распределен-

ной системы. Во-первых, потому что классическая реализация шаблона «мастер/рабочий» допускает интенсивное взаимодействие между мастерами и рабочими. На каждую запись или чтение очередь блокируется, что может привести к блокирующей конкуренции процессов. Во-вторых, управление рабочими процессами должно проходить в динамике, так как системы распределенной обработки должны быть масштабируемыми по определению. В случае статического управления добавления новых узлов в системе запрещены, так как фиксированы все маршруты.

Кроме того, основное требование к системе – обработка сбоев рабочих процессов. Система должна обнаружить ситуацию отказа и перенаправить всю работу данного рабочего процесса в соответствии с выбранным алгоритмом.

Для решения данных проблем предложенный модуль расширения использует два механизма: пакетирование работы и динамическое управление рабочими процессами, основанными на кластерных событиях Terracotta.

Для эффективного использования вычислительных мощностей процессоров необходимо уменьшать количество конфликтов блокировок, сокращающих время ожидания различных потоков. В общем случае уменьшение глубины блокировок насколько это возможно является хорошим решением. Но такая концепция в случае Terracotta не всегда дает желаемый эффект, так как эта технология рассматривает критическую секцию (например, синхронизированные блоки и методы) как единую транзакцию. Все изменения, произведенные в пределах данной транзакции, собираются в одно атомарное изменение, отсылаемое на сервер во время ее завершения. Предполагается, что сбор всех данных не слишком затратная операция по сравнению с передачей изменений на сервер.

В разработанном модуле предложено и реализовано более изящное решение, используя пакетирование. Вместо того чтобы каждую работу добавлять в канал немедленно после ее создания, работы формируются в пакет, задается некоторое пороговое значение по достижении которого весь набор работ добавляется в канал в одной транзакции. Рабочие процессы также могут использовать данную стратегию во время добавления полученных в процессе вычислений данных в канал для формирования конечного результата.

Разработанный программный модуль для построения распределенных систем создан на основе динамического управления рабочими процессами. Используя высокоуровневую систему подписки на события (кластерные события) каждый из узлов системы может подписаться или распространить событие во время подключения или отключения от разработанной распределенной системы.

Данные события представляют собой JMX-события и могут использоваться любой поддерживающий JMX библиотекой или средой.

Таким образом, используя кластерные события, организовано динамическое управление рабочими процессами и реализована возможность их добавления и отключения в режиме реального времени.

Выводы

Любая распределенная система обработки данных, особенно если она ориентирована на облачные вычисления, должна поддерживать возможность добавления и удаления ресурсов в реальном времени.

Разработанное расширение для Terracotta спроектировано, учитывая требования, предъявляемые к распределенным системам. Оно включает

- обработку отказов системы во время работы;
- обработку отказов рабочего процесса;

– динамическое управление вычислительными ресурсами системы;

– механизм пакетирования для повышения производительности системы.

Предложенный модуль представляет собой гибкую, простую в освоении и использовании среду для построения распределенных систем.

На основании проведенных экспериментов было показано, что достигается почти линейный прирост производительности при подключении новых узлов в реальном времени.

Список литературы

1. Буза М.К. Системы параллельного действия / Буза М.К. – Минск : Новое знание, 2009. – 415 с.
2. Цимбал А.А. Технология создания распределенных систем / А.А. Цимбал, М.Л. Аншина. – Спб. Питер, 2003.– 576 с.
3. Tracy Brown Collins. The Definitive Guide to Terracotta cluster the JVM™ for Spring, Hibernate and POJO Scalability. – Apress, 2008. – 365 p.
4. Топорков В.В. Модели распределенных вычислений / Топорков В.В. – М. : ФИЗМАТЛИТ, 2004. – 320 с.
5. Буза М.К. Проектирование программ для многоядерных процессоров: учебно-методическое пособие / М.К. Буза, О.М. Кондратьева. – Мн. : БГУ, 2013. – 48 с.
6. Benedict Gaster. Heterogeneous Computing with OpenCL. Elsevier. 2011. – 278 p.

References

1. Bouza M.K. Sistemy parallelnogo deistvia. Minsk, Novoe znanie, 2009. – 415 p.
2. Tsymbal A.A. Tehnologia sozdania raspredelennyh sistem / A.A. Tsymbal, .L. Anshina. – Spb. Piter, 2003.– 576 p.
3. Tracy Brown Collins. The Definitive Guide to Terracotta cluster the JVM™ for Spring, Hibernate and POJO Scalability. – Apress, 2008. – 365 p.
4. Toporkov V.V. Modeli raspredelennyh vychisleny. – M.: FIZMATLIT, 2004. – 320 p.
5. Bouza M.K. Proektirovanie programm dlya mnogoyadernyh processorov: uchebno- metodicheskoe posobie / M.K. Bouza, O.M. Kondratjeva. – Mn.: BGU, 2013. – 48 p.
6. Benedict Gaster. Heterogeneous Computing with OpenCL. Elsevier. 2011. – 278 p.

RESUME

M.K. Bouza

Environment Design of Distributed Data Processing

Reviewed and analyzed a number of existing concepts of distributed data processing. The basic requirements to such systems, and strategies for their implementation were suggested. Selected and proved technology of open source that supports distributed processing. Expanded its functional capabilities for processing large amounts of data, identification and handling failures workflows. To reduce wait times in various threads in the queue mechanism packaging works was suggested.

Статья поступила в редакцию 02.04.2014.